Algorithms - Spring '25

MSSB

Recap

- HW posted, due next Wed
  - ↳ MST & Shortest Paths
- Next week's readings are up
- Algorithms visualizer: [demo]
- End time note: please stop me if I do go over!

Shortest paths so far

$s \rightsquigarrow u \to v$

Key: relaxing tense edges $d(v) > d(u) + w(u \to v)$

Time to compute shortest path tree rooted at any $s \in V$.

- If your graph is unweighted:

  #! BFS: $O(V+E)$

- If your graph is a DAG:

  Top sort + DP: $O(V+E)$

- If no negative edges:

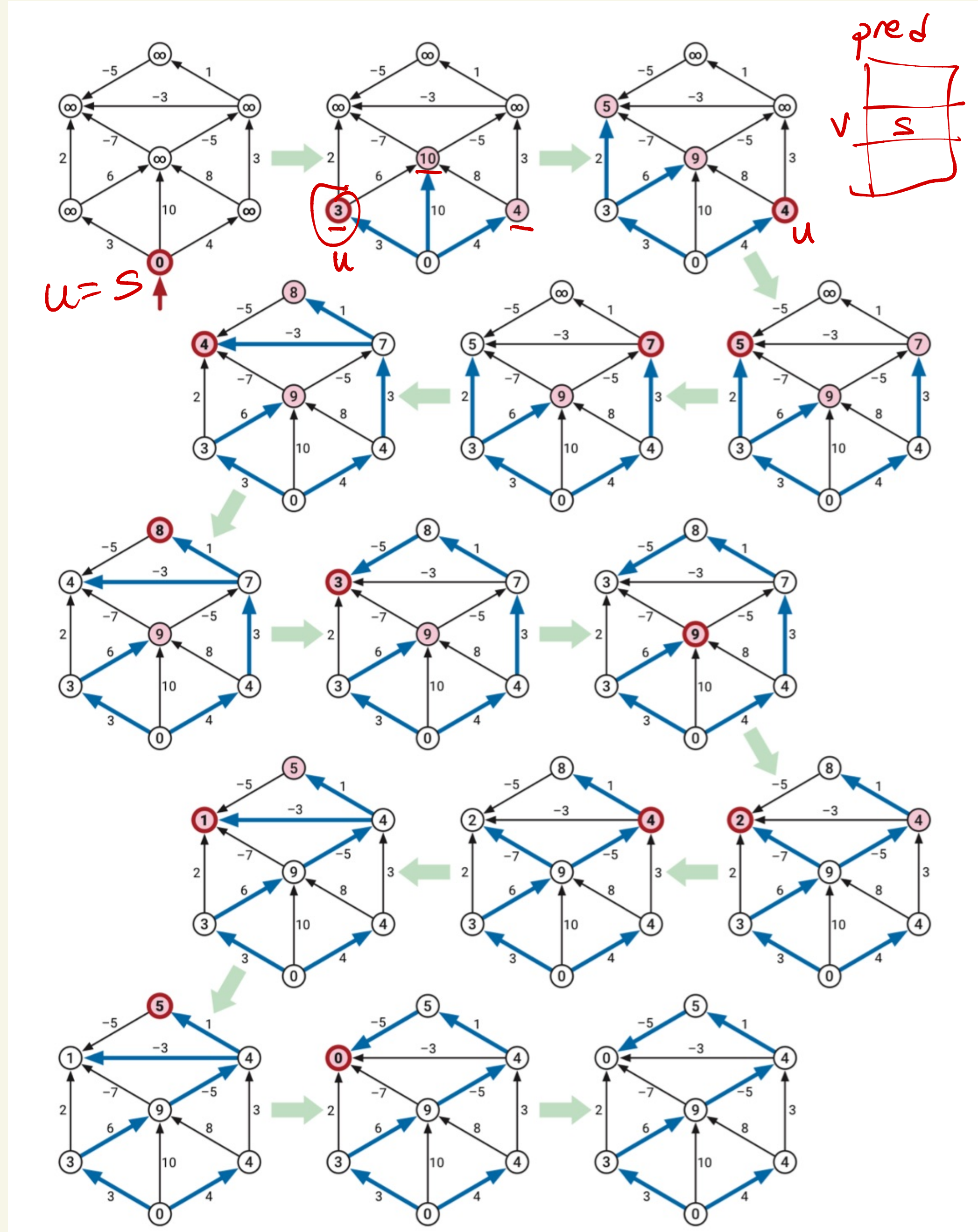
  Dijkstra: $O(E \log V)$

Dijkstra: Saw 2 versions.

One of them works on negative edges (no cycles):

```
DIJKSTRA(s):
  INITSSSP(s)
  INSERT(s, 0)
  while the priority queue is not empty
      u ← EXTRACTMIN()
      for all edges u→v
          if u→v is tense
              RELAX(u→v)
              if v is in the priority queue
                  DECREASEKEY(v, dist(v))
              else
                  INSERT(v, dist(v))
```

Figure 8.11. Dijkstra's algorithm.



worst case: exp time

If negative edges and (potentially) negative cycles:

Bellman-Ford

```
BELLMANFORD(s)
    INITSSSP(s)
    repeat V − 1 times
        for every edge u→v
            if u→v is tense
                RELAX(u→v)
    for every edge u→v
        if u→v is tense
            return "Negative cycle!"
```

Runtime:

$O(VE)$

- or -

```
BELLMANFORDFINAL(s)
    dist[s] ← 0
    for every vertex v ≠ s
        dist[v] ← ∞
    for i ← 1 to V − 1
        for every edge u→v
            if dist[v] > dist[u] + w(u→v)
                dist[v] ← dist[u] + w(u→v)
```

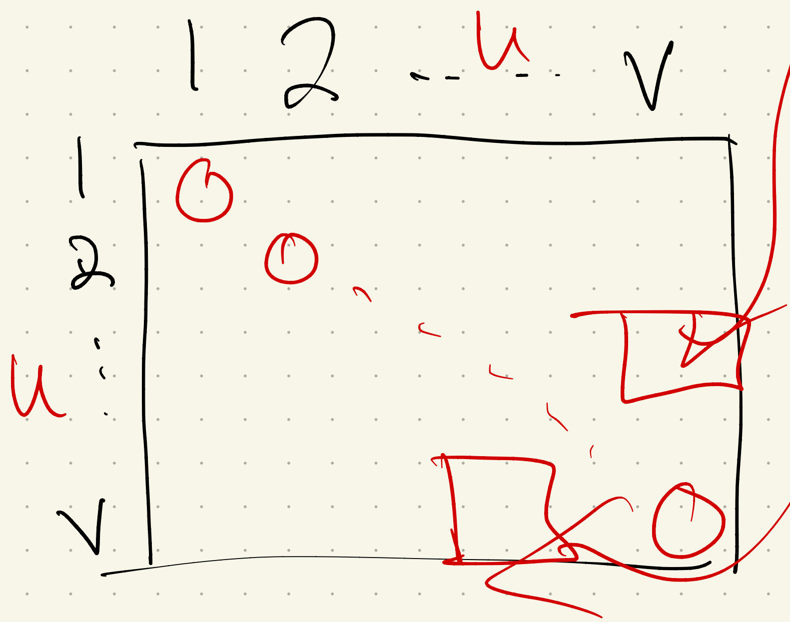
Multiple-Source Shortest Paths (MSSP)

MSSP(G):

for each pair $u, v \in G$:

- Compute shortest $u \rightsquigarrow v$
- Compute shortest $v \rightsquigarrow u$
- Store in $Dist[u,v]$ + $Dist[v,u]$

Dist:



Lookup: $O(1)$

Computing MSSPs:

First attempt: for each vertex V
Run SSSP(v)

Runtime: $O(V \times (\text{SSSP alg}))$

- if unweighted or a DAG, SSSP was $O(V+E) \Rightarrow O(V(V+E))$
- no negative edge weights, Dijkstra was $O(E \log V) \Rightarrow O(VE\log V) = O(V^3 \log V)$
- Bellman-Ford was $O(VE)$

$\Rightarrow O(V^2 \underbrace{E}_{\leq V^2}) = O(V^4)$

Side question:

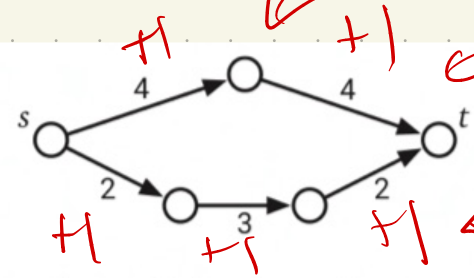
Since negative edges are bad, why can't we just re-weight?

Idea: increase all edge weights by some amount, so all positive.

Doesn't work:

2 longer

8 → 16



3 longer

Figure 9.1. Increasing all the edge weights by 2 changes the shortest path from s to t.

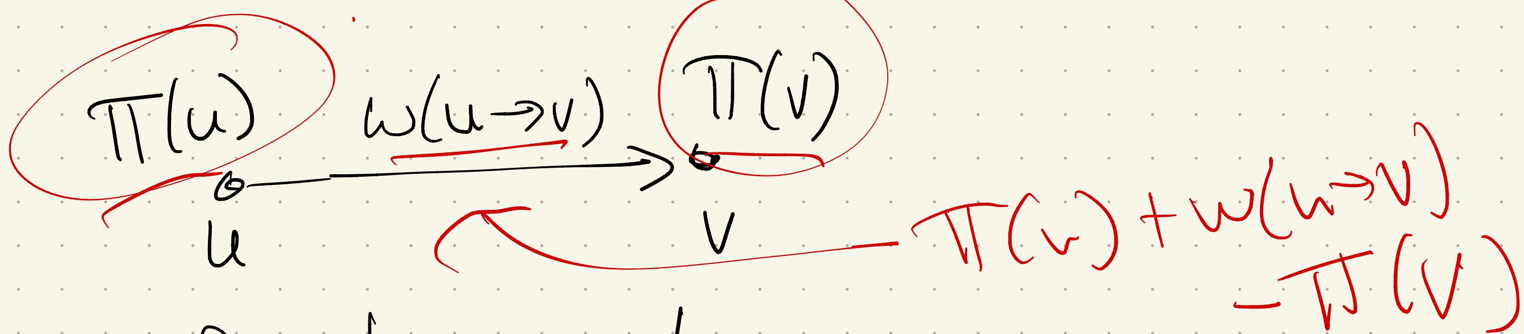
7 → 10

Why? not reweighting paths the same amount

Another idea (that works):

Suppose each v has a price attached, $\pi(v)$.

(Still have edge weights.)



Define a new function w':

$$w'(u \to v) = \pi(u) + w(u \to v) \boxed{-\pi(v)}$$

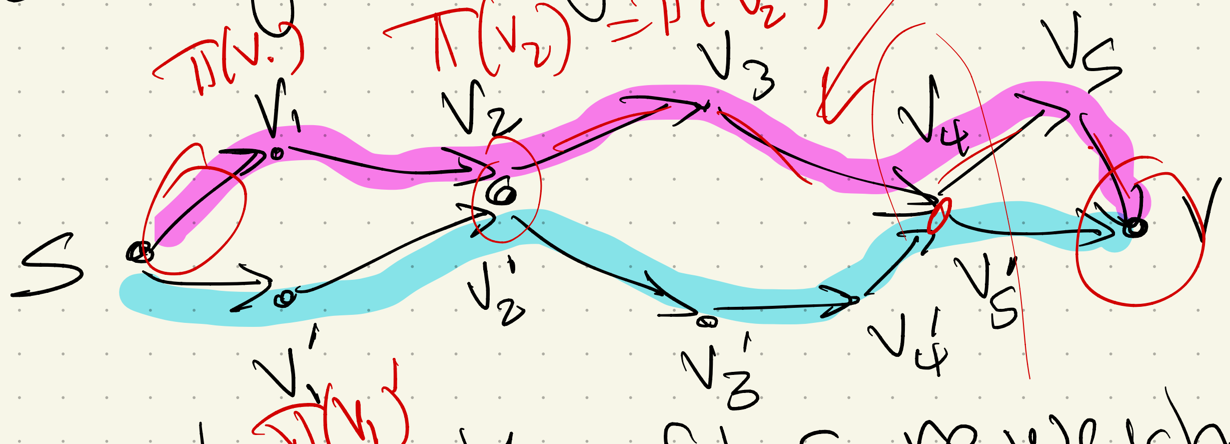
Claim: Shortest path is unchanged! all paths stay in same "order"

Claim: under w', shortest paths are the same as under w.

Why? Consider 2 $s \rightsquigarrow v$ paths:

$\pi(v_3) + w(v_3 \to v_4) - \pi(v_4)$

(in original weights)



weights: $\pi(v_i)$

purple: $w(s \to v_1) + w(v_1 \to v_2) + \dots w(v_5 \to v)$

Purple path after reweighting:

$$\left[\pi(s) + w(s \to v_1) - \pi(v_1)\right] + \left[\pi(v_1) + w(v_1 \to v_2) - \pi(v_2)\right] + \dots = \pi(s) + \text{original path} - \pi(v)$$

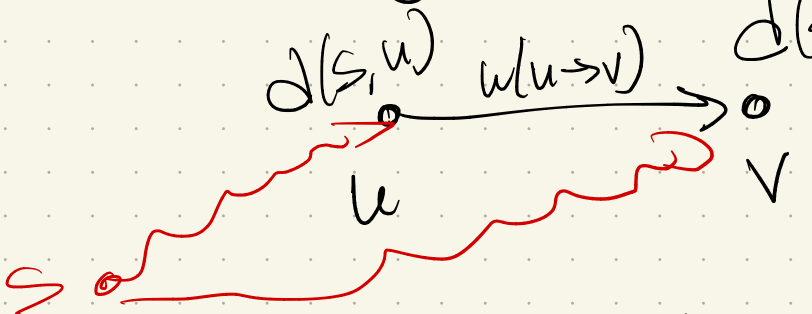
+ blue path: $\pi(s) + \text{original blue} - \pi(v)$

Johnson's algorithm for MSSP:

use this new kind of weight function!

- Run Bellman-Ford once from some $s \in V$ — $O(V \cdot E)$
  - ↳ no negative cycles or give up
  - no tense edges
- Reweight (if it works): let $\pi(v) = d(s,v)$

$$w'(u \to v) = d(s,u) + w(u \to v) - d(s,v)$$



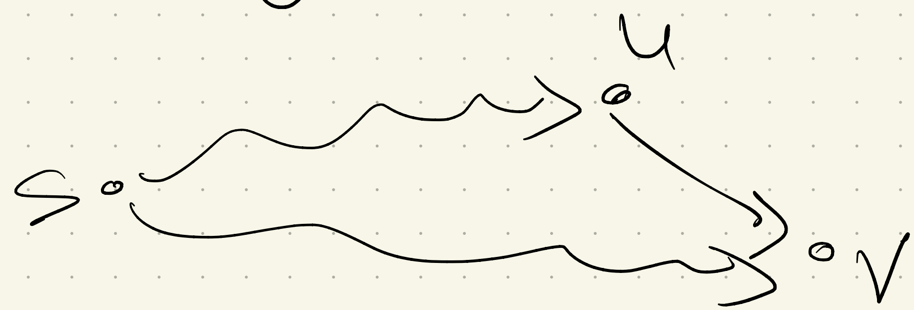
not tense

$$d(s,u) + w(u \to v) \geq d(s,v)$$

- Now, all paths are positive, so can use faster algorithm for other SSSPs!

Aside: Why positive?

After SSSP, no edges are tense:



so $d(s,u) + w(u \to v) \geq d(s,v)$

rearrange:

$$\hookrightarrow d(s,u) + w(u \to v) - d(s,v)$$
$$= w'(u \to v) \geq 0$$

So - Algorithm:

Runtime:

$O(VE\log V)$

P: $\pi(s) + (\text{path}) - \pi(v)$

min

```
JohnsonAPSP(V, E, w):
    ⟨⟨Add an artificial source⟩⟩
    add a new vertex s
    for every vertex v
        add a new edge s→v
        w(s→v) ← 0
    ⟨⟨Compute vertex prices⟩⟩
    dist[s,·] ← BellmanFord(V, E, w, s)
    if BellmanFord found a negative cycle
        fail gracefully
    ⟨⟨Reweight the edges⟩⟩
    for every edge u→v ∈ E
        w′(u→v) ← dist[s,u] + w(u→v) − dist[s,v]
    ⟨⟨Compute reweighted shortest path distances⟩⟩
    for every vertex u
        dist′[u,·] ← Dijkstra(V, E, w′, u)
    ⟨⟨Compute original shortest-path distances⟩⟩
    for every vertex u
        for every vertex v
            dist[u,v] ← dist′[u,v] − dist[s,u] + dist[s,v]
```

Figure 9.2. Johnson's all-pairs shortest paths algorithm

directed graph

$O(VE)$

reweight

$V \cdot E\log V$

Compare to naive: $O(V^2E) = O(V^4)$

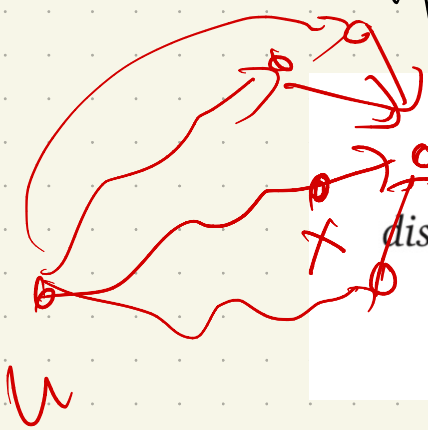
V vs $\log V$

More Improvements

Set up recursive approach:

Let $dist(u,v,\ell)$ = best $u \rightsquigarrow v$ path using at most ℓ edges.

Then, back to "keep relaxing edges" approach:



don't use another edge

$$dist(u,v,\ell) = \begin{cases} 0 & \text{if } \ell = 0 \text{ and } u = v \\ \infty & \text{if } \ell = 0 \text{ and } u \neq v \\ \min \left\{ \begin{array}{c} \boxed{dist(u,v,\ell-1)} \\ \min\limits_{x \to v} \left(dist(u,x,\ell-1) + w(x \to v) \right) \end{array} \right\} & \text{otherwise} \end{cases}$$

build length ℓ via length $\ell-1$

do use another

Code:

```
SHIMBELAPSP(V, E, w):
  for all vertices u
      for all vertices v
          if u = v
              dist[u, v, 0] ← 0
          else
              dist[u, v, 0] ← ∞

  for ℓ ← 1 to V − 1
      for all vertices u
          for all vertices v ≠ u
              dist[u, v, ℓ] ← dist[u, v, ℓ − 1]
              for all edges x→v
                  if dist[u, v, ℓ] > dist[u, x, ℓ − 1] + w(x→v)
                      dist[u, v, ℓ] ← dist[u, x, ℓ − 1] + w(x→v)
```

} paths of length 0

might need V-1 edges

Can improve

```
ALLPAIRSBELLMANFORD(V, E, w):
  for all vertices u
      for all vertices v
          if u = v
              dist[u, v] ← 0
          else
              dist[u, v] ← ∞

  for ℓ ← 1 to V − 1
      for all vertices u
          for all edges x→v
              if dist[u, v] > dist[u, x] + w(x→v)
                  dist[u, v] ← dist[u, x] + w(x→v)
```

Runtime:

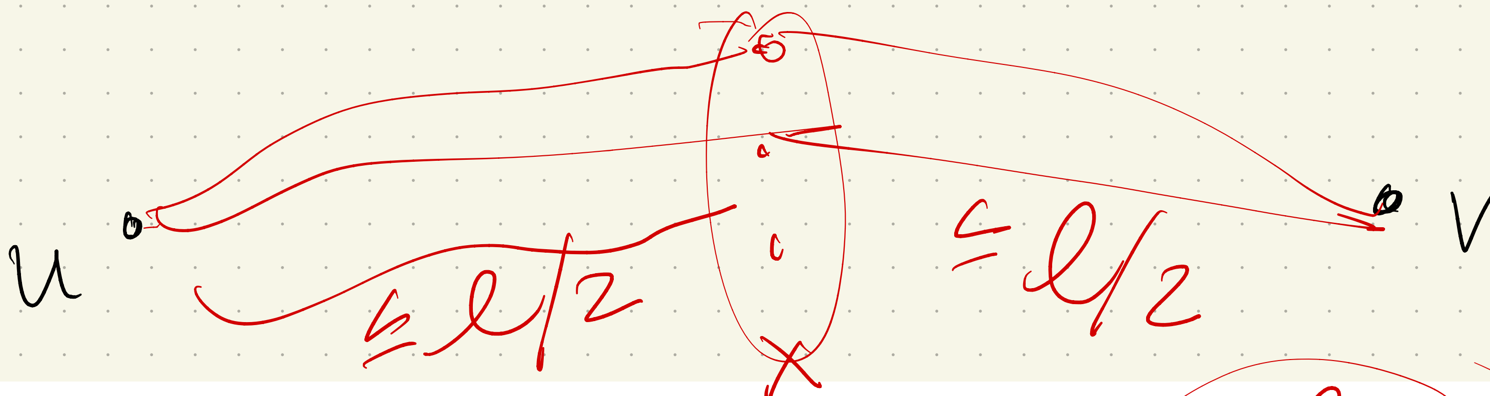
$V^2 E$

$\hookrightarrow \le V^4$

Wait → that's worse!

But, let's try a more balanced divide & conquer!

Instead of going $n-1 \to n$, can we add longer paths?



$$dist(u,v,\ell) = \begin{cases} w(u\to v) & \text{if } \ell = 1 \\ \min\limits_{x}\left(dist(u,x,\ell/2) + dist(x,v,\ell/2)\right) & \text{otherwise} \end{cases}$$

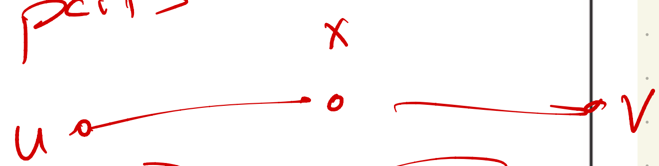
best $u \to v$ path of length $\le \ell$

Code:

```
FischerMeyerAPSP(V, E, w):
  for all vertices u
    for all vertices v
      dist[u, v, 0] ← w(u→v)

  for i ← 1 to ⌈lg V⌉          ⟨⟨ℓ = 2^i⟩⟩
    for all vertices u
      for all vertices v
        dist[u, v, i] ← ∞
        for all vertices x
          if dist[u, v, i] > dist[u, x, i − 1] + dist[x, v, i − 1]
            dist[u, v, i] ← dist[u, x, i − 1] + dist[x, v, i − 1]
```

all pairs



V^2

Can also save space

```
LeyzorekAPSP(V, E, w):
  for all vertices u
    for all vertices v
      dist[u, v] ← w(u→v)

  for i ← 1 to ⌈lg V⌉          ⟨⟨ℓ = 2^i⟩⟩
    for all vertices u
      for all vertices v
        for all vertices x
          if dist[u, v] > dist[u, x] + dist[x, v]
            dist[u, v] ← dist[u, x] + dist[x, v]
```

$V^3 \log V$

Can we get better?

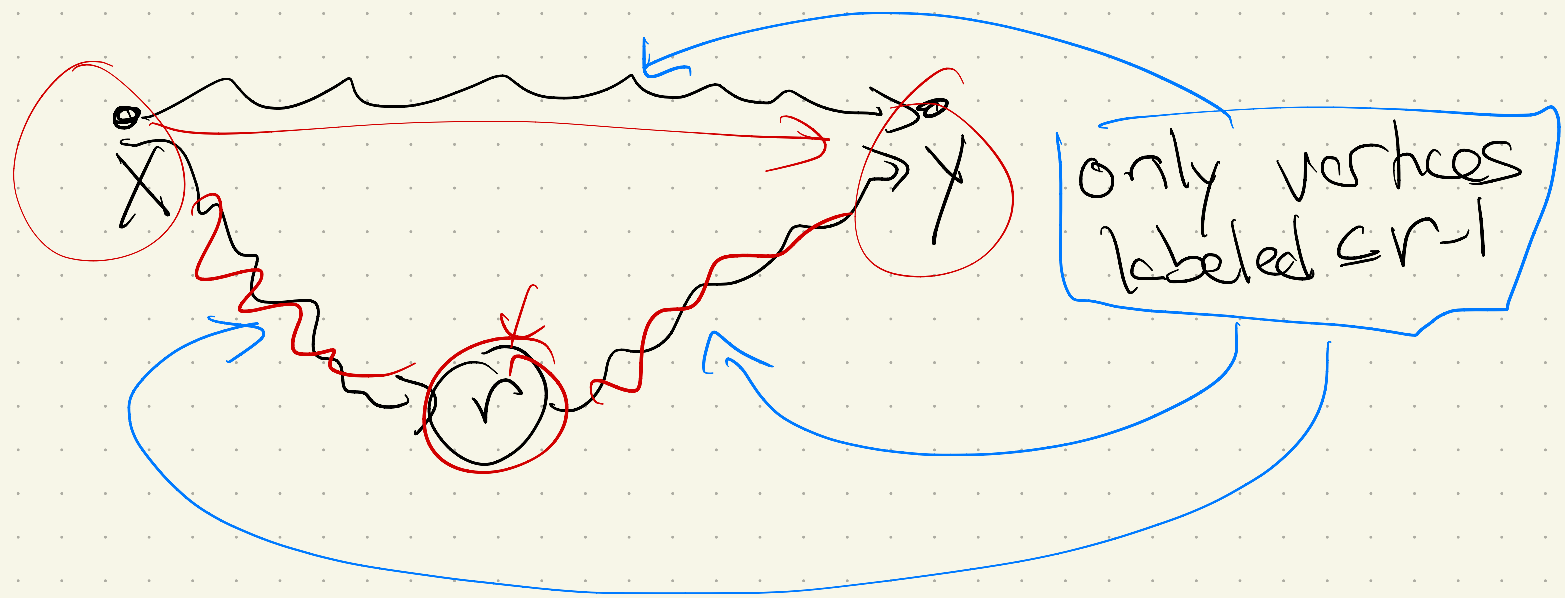
Floyd-Warshall: instead of path length does vertex order

order vertices $1, \dots, V$

Let $d(x, y, r) =$

best length path via vertices labeled $1 \dots r$

Then:



Recursion:

$$dist(u,v,r) = \begin{cases} w(u \to v) & \text{if } r = 0 \\ \min \left\{ \begin{array}{c} dist(u,v,r-1) \\ dist(u,r,r-1) + dist(r,v,r-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

use vertex r

don't use vertex r

code

```
KLEENEAPSP(V, E, w):
  for all vertices u
      for all vertices v
          dist[u, v, 0] ← w(u→v)

  for r ← 1 to V
      for all vertices u
          for all vertices v
              if dist[u, v, r − 1] < dist[u, r, r − 1] + dist[r, v, r − 1]
                  dist[u, v, r] ← dist[u, v, r − 1]
              else
                  dist[u, v, r] ← dist[u, r, r − 1] + dist[r, v, r − 1]
```

Runtime:

save space

```
FLOYDWARSHALL(V, E, w):
  for all vertices u
      for all vertices v
          dist[u, v] ← w(u→v)

  for all vertices r
      for all vertices u
          for all vertices v
              if dist[u, v] > dist[u, r] + dist[r, v]
                  dist[u, v] ← dist[u, r] + dist[r, v]
```

Next topic: Flows & cuts